

REMARKS

Claims 1-20 are currently pending in the present application. No claims have been amended in this response. Reconsideration of the claims is respectfully requested; for the convenience of the examiner, a complete clean copy of the claims is included as an appendix to this response.

I. Summary of Present Invention

The present invention is a method for deferred deletion of entries in a directory service backing store. Although shown as a preferred embodiment within the specification, the invention is not limited to a Lightweight Directory Access Protocol (LDAP) directory service provided with a DB/2 backing store. As stated in the specification, the principles of the present invention may be practiced in other types of directory services, e.g., X.500, and using other relational database management systems, e.g., Oracle, Sybase, Informix, etc., as the backing store.

In either the present invention or the prior art, an entry in an LDAP directory is deleted using an SQL statement. In the prior art, the directory server responds to the delete entry statement by instituting a global lock on the database tables to ensure that data in those tables cannot be modified while the entry is being deleted from the directory. In contrast, the present invention provides an enhanced delete operation whereby the entry is marked for deletion, and the actual deletion is completed at a later time.

More specifically, the present invention is a method for deleting an entry from a directory in which directory information is stored in a set of database tables; the deletion is initiated in response to a request to delete a directory entry. In response, the directory entry is tagged in some manner as being a deleted entry, preferably by setting the entry's creation time to a null value. If a search query is received thereafter, the method excludes tagged entries from search results that would otherwise satisfy the search query. At a periodic interval, the routine then periodically searches for tagged entries, and references to the tagged entries are then deleted throughout the set of database tables. In this manner, the completion of the entry deletion operation is deferred to enable directory queries to be processed even if deleted entries have not yet been fully expunged from the directory.

II. 35 U.S.C. § 102(e)—Anticipation—*Bachmann et al.*

The Office action has rejected claims 14-20 under 35 U.S.C. § 102(e) as being anticipated by *Bachmann et al.*, “Method of hierarchical LDAP searching with relational tables”, U.S. Patent No. 6,085,188, issued 04/04/2000 (hereinafter *Bachmann et al.*). This rejection is respectfully traversed.

Bachmann et al. is directed to a method and system for hierarchical LDAP searching in an LDAP directory service having a relational database management system (RDBMS) as a backing store. In columns 3 and 4 of *Bachmann et al.*, an explanation of an LDAP directory service is provided as well as an explanation of implementing the LDAP directory service using an RDBMS. In a similar manner, pages 9-14 of the present application also provide an explanation of an LDAP directory service as well as an explanation of implementing the LDAP directory service using an RDBMS. As such, *Bachmann et al.* and the present application share a common infrastructure of an LDAP directory service having an RDBMS backing store.

However, *Bachmann et al.* does not disclose the claimed features of the present patent application, and this should be apparent to one of ordinary skill in the art after a careful reading of the present patent application. Although *Bachmann et al.* teaches LDAP table structures in a manner similar to the present invention, *Bachmann et al.* does not teach the features of the claimed invention in the present application. At most, *Bachmann et al.* has a common assignee with the present patent application, and *Bachmann et al.* also has a common co-inventor with the present patent application, David Bachmann; these facts are irrelevant to the determination of patentability. For example, independent claim 14 of the present application, which is one of three independent claims that are rejected as anticipated by *Bachmann et al.*, reads as follows:

14. A method for searching a database from a directory service, comprising the steps of:
 - responsive to a search for directory entries that satisfy a search query, excluding given entries from search results that otherwise satisfy the search query, wherein the given entries identify database entries that have been tagged for deletion; and
 - returning the search results.

Bachmann et al. does not disclose “excluding given entries from search results that otherwise satisfy the search query” nor “wherein the given entries identify database entries that have been tagged for deletion”. In fact, the only portion of *Bachmann et al.* related to a deletion operation is not applied by the current Office action to independent claim 14.

This 102(e) rejection includes several citations to portions of Bachmann et al., none of which disclose the claimed features. The abstract of Bachmann et al. summarizes that the invention in Bachmann et al. is directed to a manner of mapping directory entries in a naming hierarchy into first and second relational tables that are used to filter lists of entries within a given search scope for evaluation. The rejection of claims 14-20 merely applies portions of the LDAP search mechanism disclosed in Bachmann et al. against the claimed features of the present application without careful consideration of the claim language and the subject matter

Against the first feature of independent claim 14, the rejection cites column 6, lines 14-25:

According to the invention, the relation tables model the relationship between the LDAP entries to facilitate one level and subtree searches without recursive queries. In both cases, the search begins by going into the database and using the LDAP filter criteria to retrieve a list of entries matching the filter criteria. If the search is a one level search, the parent table is then used to filter out EIDs that are outside the search scope (based on the starting point or base DN). Likewise, if the search is a subtree search, the descendant table is then used to filter out EIDs that are outside the search scope (again, based on the base DN). Generally, the tables are not required to be used in a base level search.

An EID is a unique identifier associated with each directory entry; it should be obvious that the above cited portion does not disclose “excluding given entries from search results that otherwise satisfy the search query” because the EIDs that are “filtered out” from the search do not satisfy the search query. Moreover, it should be obvious that the above cited portion does not disclose “wherein the given entries identify database entries that have been tagged for deletion”; the mechanism of tagging entries for deletion is not present within Bachmann et al..

Against the second feature of independent claim 14, i.e. “returning the search results”, the rejection cites several portions of Bachmann et al.. Clearly, both the present application and the system disclosed in Bachmann et al. concern searching directories and “returning the search results”.

With respect to dependent claim 15, the present invention and the system disclosed in Bachmann et al. both clearly concern LDAP directory services.

With respect to independent claim 16, this claim is similar to independent claim 1 in that claim 16 is directed to a computer-readable medium comprising instructions to accomplish the method of claim 1. Hence, it is unclear why the Office action rejected claim 1 as being anticipated by Barrett et al. and then rejected claim 16 as being anticipated by Bachmann et al..

Against the first feature and last feature of independent claim 16, the rejection cites column 6, lines 43-59, which reads:

Various routines are provided for manipulating entries and for searching using the relational tables described above. As described below, the ldap_entry table includes the parent table, and the descendant table is the ldap_desc table. These routines are now described generally.

FIG. 7 is a flowchart illustrating an ldap_delete (or delete) routine that removes entries from the database. It begins at step 50. At step 52, the routine maps the distinguished name (DN) to the entry identifier (EID). The routine then continues at step 54 by obtaining the ancestor (PEID's) from the ldap_desc table. For each PEID retrieved, the routine then performs a processing loop at step 56. In particular, the routine removes the PEID and EID pair from the descendant table (ldap_desc table), at step 58, and then cycles. When step 56 is complete (i.e. all PEIDs have been processed), the routine branches to step 60 to remove the EID entry from the ldap_entry table. This completes the processing.

This portion of Bachmann et al. is the only portion of the reference that discusses or mentions a deletion operation; the portion describes a method for stepping through the relational database tables that support the LDAP directory entries and deleting an entry. The cited portion does not mention or discuss anything close to the features of “means responsive to a request to delete a directory entry for tagging the directory entry in a first table” and “ means for deleting references to the tagged entries throughout the set of database tables”.

Against the second feature of independent claim 16, the rejection cites column 5, line 51, to column 6, line 13, which reads:

The invention thus implements two relations or tables in order to support LDAP search: parent/child and ancestor/descendants. In the parent table, the EID field is the unique identifier of an entry in the LDAP naming hierarchy. The PEID field is the unique identifier of the parent entry in the naming hierarchy. In the descendant table, the AEID field is the unique identifier of an ancestor LDAP entry in the LDAP naming hierarchy. The DEID field is the unique identifier of the descendent LDAP entry.

LDAP provides a number of known functions including query (search and compare), update, authentication and others. The search and compare operations are used to retrieve information from the database. For the search function, the criteria of the search is specified in a search filter. The search filter typically is a Boolean expression that consists of attribute name, attribute value and Boolean operations like AND, OR and NOT. Users can use the filter to perform complex search operations. The filter syntax is defined in RFC 1960 (RFC stands for Request For Comments, See <http://www.cis.ohiostate.edu/hypertext/information/rfc.html>).

In addition to the search filter, users can also specify where in the directory tree structure the search is to start. The starting point is called the base DN. The search can be

applied to a single entry (a base level search), an entry's children (a one level search), or an entire subtree (a subtree search). Thus, the "scope" supported by an LDAP search are: base, one level and subtree. LDAP does not support search for arbitrary tree levels and path enumeration.

The cited portion does not mention or discuss anything even remotely similar to the features of "means for periodically searching for tagged entries in the first table during a cleanup process interval".

With respect to dependent claim 17, this claim is similar to the first element of independent claim 14 in that claim 17 is directed to a computer-readable medium comprising instructions to accomplish the first step of the method of claim 14, i.e. "excluding given entries from search results that otherwise satisfy the search query". Again, this feature is not shown in Bachmann et al.

With respect to dependent claim 18, the present invention and the system disclosed in Bachmann et al. both clearly concern LDAP directory services.

With respect to independent claim 19, this claim is similar to independent claim 1 and its dependent claim 2 in that claim 19 is directed to a computer-readable medium comprising instructions to accomplish a method combining the elements of claims 1 and 2. Hence, it is unclear why the Office action rejected claims 1 and 2 as being anticipated by Barrett et al. and then rejected claim 19 as being anticipated by Bachmann et al. The elements of independent claim 19 are similar to the elements of claims 16 and 17 and are not shown in Bachmann et al., as previously shown above.

With respect to dependent claim 20, the present invention and the system disclosed in Bachmann et al. both concern LDAP directory services.

Clearly, the rejection has not carefully considered the elements of the claims nor has the rejection pointed out the claimed features within Bachmann et al. as is required for a proper anticipation rejection. As stated at MPEP § 2131: "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). "The identical invention must be shown in as complete detail as is contained in the ... claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Hence, the rejection of claims 14-20 is improper, and Applicant requests that the rejection be withdrawn.

III. 35 U.S.C. § 102(e)—Anticipation—*Barrett et al.*

The Office action has rejected claims 1-5 and 9-11 under 35 U.S.C. § 102(e) as being anticipated by Barrett et al., “Method and system for traversing linked list record based upon write-once predetermined bit value of secondary pointers”, U.S. Patent No. 5,247,658, issued 09/21/1993 (hereinafter Barrett et al.). This rejection is respectfully traversed.

Review of the teachings of the *Barrett et al.* reference

As noted in the background section of the reference, information on computer systems is generally organized into files in which a file is a collection of information. File systems typically organize files using directory structures on a storage device; directories typically organize files in logical, hierarchical manner. Barrett et al. is directed to a method and system for storing files on a computer file storage device. As stated in the abstract:

The files are organized into a hierarchical directory structure. The directory structure comprises directory entries and file entries. The file entries and directory each contain a primary and a secondary pointer. The secondary pointer is initially set to a predefined value. When an entry is to be updated, the secondary pointer is overridden with a value that points to the superseding entry. This directory structure is especially suitable to be used in a write-once computer memory.

The methodology disclosed in Barrett et al. is particularly designed for storing and updating files on a write-once, read-memory flash memory. A linked list data structure is used to link “extents” that compose a file. Because of the write restrictions on this type of memory, the file system adds directory records, file records, extent records, and extents by allocating space for the extents as needed starting from address zero in the memory device. The file system treats the memory device as a stack-like device; data is pushed onto the stack to effect allocation, but data is never popped from the stack. From another perspective, data is allocated from the free memory space but never unallocated. When data storage is needed for a file, a primary pointer is used to point to an extent that has been allocated for the newly added file storage space; once allocated, the entry cannot be unallocated because the information in the read-only memory can only be cleared by erasing the entire memory. The system uses the secondary pointer to indicate that data stored in the file system data structure has been superseded or updated; the secondary pointer then points to a record/entry that contains the

superseding data because data can be truly deleted or overwritten. A directory or file entry is never truly deleted because it cannot be physically removed from the linked list; a directory or file record is marked as “deleted” from the linked list by logically clearing the exist/delete bit of the status byte of the directory or file entry to show that the entry does not contain valid content information and should only be used for retrieving a pointer to another entry in the linked list.

Review of the rejection of claims 1-5 and 9-11

As an important preliminary issue, it should be noted that, unfortunately, the term “directory” has multiple meanings; the “directory” terminology that is used by Barrett et al. has a different meaning than the “directory” terminology that is used by the present invention. A printout of the definitions of “directory”, “directory service”, and “LDAP” are included at the end of this response; these definitions were obtained from the TechWeb.com Web site, which contains a technology encyclopedia called “TechEncyclopedia”. TechWeb defines “directory” with three different meanings, each of which is a newer use of concept for “directory” than the previous meaning:

- (1) A simulated file folder on disk. Programs and data for each application are typically kept in a separate directory (spreadsheets, word processing, etc.). Directories create the illusion of compartments, but are actually indexes to the files which may be scattered all over the disk. UNIX and DOS use the term directory, while the Mac and Windows use the term “folder”;
- (2) A database of users, hardware devices and applications in a network. See DSML, directory service and metadirectory;
- (3) A search site on the Web that catalogs Web sites by subject and also manually indexes the site, providing a brief description of its content. Yahoo! is the most well-known directory site. See Web search sites, metasearch sites and Yahoo!.

Barrett et al. uses the term “directory” in the first, decades-old meaning, whereas the present invention uses the second meaning, which has been in use since approximately 1987-1988. Although both uses of “directory” can be associated with storing data in a hierarchical fashion using “entries”, the two meanings are different. Barrett et al. is directed to a methodology for hierarchically organizing files using a directory, whereas the present invention is directed to a methodology for deleting entries within a directory-type database.

As explained in more detail in the specification of the present application, a directory is like a database, but tends to contain more descriptive, attribute-based information. The LDAP directory service model is based on entries. An entry is a collection of attributes that has a name, called a distinguished name (DN). The DN is used to refer to the entry unambiguously. Each of the entry's attributes has a type and one or more values. The types are typically mnemonic strings, like "cn" for common name or "mail" for email address. The values depend on what type of attribute it is. For example, a mail attribute might contain the value "jdoe@uspto.gov", whereas a "jpegPhoto" attribute would contain a photograph in binary JPEG format. In LDAP, directory entries are arranged in a hierarchical tree-like structure that reflects political, geographic, and/or organizational boundaries. For example, entries representing countries may appear at the top of the tree, while below the top entries are entries representing states or national organizations, followed by entries representing people, organizational units, printers, or documents.

LDAP defines operations for interrogating and updating the directory. Operations are provided for adding and deleting an entry from the directory, changing an existing entry, and changing the name of an entry. Most of the time, though, LDAP is used to search for information in the directory. The LDAP search operation allows some portion of the directory to be searched for entries that match some criteria specified by a search filter. Information can be requested from each entry that matches the criteria. For example, one may want to search the entire directory subtree below the root of a corporation directory for employees with the name J. Doe, retrieving the email address of each entry found; LDAP lets an application perform this function easily and in accordance with open standards.

Given the different meanings of "directory" and the fact that the Barrett et al. reference uses a first meaning while the present application uses a second meaning, Applicant contends that the two meanings for "directory" have been confused in this rejection of claims 1-5 and 9-11 and in the following rejection of claims 6-8, 12, and 13. The rejection applies an improper interpretation to the teachings of the Barrett et al. reference because of this terminology discrepancy, and Applicant requests that the rejection be withdrawn on this issue alone.

Moving from this preliminary issue, Barrett et al. does not disclose the claimed features of the present patent application, which should be apparent to one of ordinary skill in the art after a careful comparison of the present patent application with Barrett et al.. The following arguments emphasize that

the teachings of Barrett et al. are directed to subject matter that is not at all similar to the present invention.

With respect to independent claim 1, the rejection cites one portion of Barrett et al..

Independent claim 1 reads as follows:

1. A method for deleting entries from a directory in which directory information is stored in a set of database tables, comprising the steps of:
responsive to a request to delete a directory entry, tagging the directory entry in a first table;
periodically searching for tagged entries in the first table during a cleanup process interval; and
deleting references to the tagged entries throughout the set of database tables.

Against the totality of independent claim 1, the rejection cites Figure 18 and column 11, lines 33-39, which read:

FIG. 18 shows a flow diagram of a routine that deletes a directory or file from the FEProm. This routine clears the exist/delete bit in the FEDirEntry. In block 1801, the system locates the directory or file to be deleted and sets the pointer D to contain the address of the directory or file. In block 1802, the system sets D-->status to indicate that the directory or file is deleted and the routine completes.

As noted several paragraphs above, a directory or file entry is never truly deleted because it cannot be physically removed from the linked list; a directory or file record is marked as “deleted” from the linked list by logically clearing the exist/delete bit of the status byte of the directory or file entry to show that the entry does not contain valid content information and should only be used for retrieving a pointer to another entry in the linked list. The portion of Barrett et al. that is applied against claim 1 refers to this functionality within the system.

Putting aside the fact that the meaning of “directory entry” differs between Barrett et al. and the present invention, a logical and consistent interpretation of this portion of Barrett et al. could only be applied against the first element of claim 1, i.e. “responsive to a request to delete a directory entry, tagging the directory entry in a first table”, wherein the step of clearing an exist/delete bit in the status byte of a directory entry in the system disclosed in Barrett et al. is analogous to the step of tagging a directory entry. In the system disclosed in Barrett et al., the act of clearing the exist/delete bit completes a “delete” operation; it is not possible physically remove or overwrite an entry as would be understood

in the standard use and interpretation of the word “delete”. With respect to the “delete” operation of a directory entry, no other subsequent operations are necessary.

In contrast, the present invention completes a delete operation as is commonly understood by one having ordinary skill in the art. More specifically, the present invention is directed to a “deferred” delete operation, as is clearly disclosed in the specification of the present invention. At some point in time after a directory entry is tagged, the deletion process is completed by searching for tagged directory entries during a cleanup interval, and any directories that are found to be tagged are then truly deleted by being physically removed or erased from the directory structure.

Barrett et al. does not disclose the second and third elements of independent claim 1 because it is not possible to delete a single, tagged directory entry without physically erasing the entire memory medium. Hence, there is no reason for the system disclosed in Barrett et al. to search for directory entries of any kind in an attempt to delete those entries because the entries cannot be truly deleted. It should be very clear to one of ordinary skill in the art that, in any interpretation of Barrett et al., that Barrett et al. does not disclose the second and third elements of claim 1.

Against dependent claim 2, which states that “the directory entry is tagged by setting its creation time to a given value”, the rejection cites a portion of Barrett et al. at column 8, lines 10-24, which reads:

In block 401 of FIG. 4, the system locates directory “P” by following the path from the root directory and setting variable P to the address of “P” directory. In block 402, the system allocates a new directory entry of the record type FEDirEntry for directory “C.” The system sets the variable C to the variable first_unallocated and allocates the space by incrementing the variable first_unallocated by the size of an FEDirEntry record. FIGS. 7A and 7B show the address space before allocation and after allocation of the FEDirEntry record, respectively. In block 403, the system sets the variables name, time, date, and attribute in the newly allocated record. In block 404, the system sets the status bit to indicate that the newly allocated entry is a directory entry, rather than a file entry.

As should be clearly apparent, the cited portion of Barrett et al. merely shows that a directory entry is timestamped to show its time of creation. It should be clear to one of ordinary skill in the art that the act of setting the creation timestamp in this manner does not disclose the act of setting the timestamp to a certain value to mark the entry as deleted. Moreover, the rejection of dependent claim 2 uses an interpretation of Barrett et al. that is inconsistent with the interpretation of Barrett et al. that is applied

against independent claim 1. The system disclosed in Barrett et al. is entirely reliant on the exist/delete bit operating in a certain manner, and other data items within the directory entry are not used to mark the directory entry as deleted.

Against dependent claims 3 and 11, which states that “the given value is a null value”, the rejection cites a portion of Barrett et al. at column 7, lines 23-34, which reads:

In a preferred embodiment, the first_unallocated pointer is not stored on the FEProm. When a FEProm is first connected to the computer (i.e., put on-line), the file system searches the FEProm from the highest address location to the lowest address for the first occurrence of a non-FNULL byte. The next higher address is the start of the unallocated area (first_unallocated). The file system should ensure that the data stored at the end of the allocated portion does not contain a FNULL. This can be accomplished by adding a non-FNULL byte to the end of any record or extent that ends in an FNULL.

As should be clearly apparent, the cited portion of Barrett et al. merely shows that the content information that is stored within an extent is not allowed to end with a byte that has an FNULL value. It should be clear to one of ordinary skill in the art that the act of adding a non-FNULL byte to a directory entry in this manner does not disclose the act of setting the creation timestamp to a null value in order to mark the directory entry as deleted.

Against the first element of dependent claim 4, which comprises the element of “performing a search for directory entries that satisfy a search query”, the rejection cites a portion of Barrett et al. at column 11, lines 49-66, which reads:

FIG. 19 is a flow diagram of a preferred subroutine that implements the changing of a file name. (The subroutine for changing a directory is similar, except that there are no associated extents.) The input parameters to this routine are the pathname of the file and the new file name. In block 1901, the system searches through the directory structure and locates the file whose name is to be changed and sets the variable P to point to the FEFileEntry. In block 1902 and 1903, the system searches for the last FEFileEntry in the linked list of entries for the file. A file will have an entry for each name change. In block 1902, if P-->secondary_ptr equals FNULL, then P points to the end of the linked list and the system continues at block 1904, else P does not point to the end of the linked list and the system continues at block 1903. In block 1903, the system sets P equal to P-->secondary_ptr to walk through the linked list.

As should be apparent, the cited portion of Barrett et al. merely shows that the linked list can be walked to find a matching file name in order to change the file name. Against the second element of dependent

claim 4, which comprises the element of “excluding tagged entries from search results that otherwise satisfy the search query”, the rejection cites a portion of Barrett et al. at column 12, lines 23-37, which reads:

FIGS. 21A and 21B show a flow diagram of a routine that changes the attribute data of a file. The input parameters are the pathname and the attribute data. In block 2101, the system searches through the directory structure to locate the file and sets the variable P to point to the FEFileEntry. In block 2102 and 2103, the system searches for the last FEFileEntry in the linked list of entries for the file. A file will have an entry for each name change. In block 2102, if P-->secondary_ptr equals FNNULL, then P points to the end of the linked list and the system continues at block 2104, else P does not point to the end of the linked list and the system continues at block 2103. In block 2103, the system sets P equal to P-->secondary_ptr to walk through the linked list..

As should be apparent, the cited portion of Barrett et al. merely shows that a linked list can be walked in order to add a new FEFileEntry entry onto the linked list of FEFileEntry entries so that the attribute data associated with a file may be changed. It should be clear to one of ordinary skill in the art that the act of determining whether or not a pointer points to the end of the linked list in this manner does not disclose the act of excluding tagged directory entries from the search results that are generated from the directory search if the tagged directory entry would otherwise satisfy a search query except for the fact that the directory entry is tagged.

Against dependent claim 5, which states that “the step of excluding tagged entries includes modifying an SQL query to exclude rows having a null change creation”, the rejection cites a portion of Barrett et al. at column 1, lines 30-40, which reads:

The information stored on nonvolatile storage devices is generally organized into files. A file is a collection of related information. Over the course of time, a computer system can store hundreds and thousands of files on a storage device, depending upon the capacity of the device. In addition to storing the information, the computer system will typically read, modify, and delete the information in the files. It is important that the computer system organize the files on the storage device so that the storing, reading, modifying, and deleting can be accomplished efficiently.

It is entirely unclear how this passage discloses anything remotely related to the modification of an SQL query.

With respect to independent claim 9, this claim is similar to independent claim 1 and dependent claim 4 in that claim 9 is a combination of the elements of claims 1 and 4. The rejection merely cites the

same portions of Barrett et al. against claim 9 that were cited against claims 1 and 4. Applicant has refuted these arguments above with respect to claims 1 and 4 and maintains that independent claim 9 is not anticipated by Barrett et al. for the same reasons.

With respect to dependent claim 10, this claim is similar to dependent claim 2. The rejection merely cites the same portion of Barrett et al. against claim 10 that was cited against claim 2. Applicant has refuted this argument above with respect to claim 2 and maintains that dependent claim 10 is not anticipated by Barrett et al. for the same reasons.

Clearly, the rejection has not carefully considered the elements of the claims nor has the rejection pointed out the claimed features within Barrett et al. as is required for a proper anticipation rejection. As stated at MPEP § 2131: “A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference.” *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). “The identical invention must be shown in as complete detail as is contained in the ... claim.” *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Hence, the rejection of claims 1-5 and 9-11 is improper, and Applicant requests that the rejection be withdrawn.

IV. 35 U.S.C. § 103(a)—Obviousness—*Barrett et al.* in view of *Bachmann et al.*

The Office action has rejected claims 6-8, 12, and 13 under 35 U.S.C. § 103(a) as unpatentable over Barrett et al. in view of Bachmann et al.. This rejection is respectfully traversed.

With respect to dependent claims 6-8, 12, and 13, each of these claims recite some aspect of an LDAP directory service or a database table that is used as a directory entry table for storing attributes. The rejection cites portions of Bachmann et al. that show the claimed features; Applicant does not dispute that Bachmann et al. discloses the elements for which the rejection relies upon Bachmann et al. as disclosing.

However, the rejection then attempts to combine the teachings of Bachmann et al. concerning directories and directory entry tables with the teachings of Barrett et al. concerning file directories. As explained in detail above, the subject matter of these references are actually concerned with different types of directories--the rejection confuses two different meanings of the term “directory”, as should be

clearly apparent to one having ordinary skill in the art. By stating “Barret [sic] does not explicitly disclose ‘the directory is a Lightweight Directory Access Protocol (LDAP) directory service and the database tables are managed by a relational database management service’”, the rejection is implicitly admitting to the wide differences between the two references.

Applicant argues that the teachings of the references cannot be combined to reject the present invention. Barrett et al. is directed to implementing a file system using particular data structures for the file directories, and a directory within a directory service is unsuitable for this purpose. Moreover, any attempt to implement a directory for a directory service within the system disclosed in Barrett et al., even if at all possible, would completely change its method of operation.

Because Barrett et al. and Bachmann et al. cannot be combined, the rejection of claims 6-8, 12, and 13 fails to present a *prima facie* case of obviousness. Applicant is supported in this argument by the MPEP, which states the following within MPEP § 2143.02:

If the proposed modification would render the prior art invention being modified unsatisfactory for its intended purpose, then there is no suggestion or motivation to make the proposed modification. *In re Gordon*, 733 F.2d 900, 221 USPQ 1125 (Fed. Cir. 1984).

If the proposed modification or combination of the prior art would change the principle of operation of the prior art invention being modified, then the teachings of the references are not sufficient to render the claims *prima facie* obvious. *In re Ratti*, 270 F.2d 810, 123 USPQ 349 (CCPA 1959).

Examiner bears the burden of establishing a *prima facie* case of obviousness.

The examiner bears the burden of establishing a *prima facie* case of obviousness based on the prior art when rejecting claims under 35 U.S.C. § 103. *In re Fritch*, 972 F.2d 1260, 23 U.S.P.Q.2d 1780 (Fed. Cir. 1992). Only when a *prima facie* case of obviousness is established does the burden shift to the applicant to produce evidence of nonobviousness. *In re Oetiker*, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); *In re Rijckaert*, 9 F.3d 1531, 1532, 28 U.S.P.Q.2d 1955, 1956 (Fed. Cir. 1993). If the Patent Office does not produce a *prima facie* case of unpatentability, then without more the applicant is entitled to grant of a patent. *In re Oetiker*, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); *In re Grabiak*, 769 F.2d 729, 733, 226 U.S.P.Q. 870, 873 (Fed. Cir. 1985). In response to an assertion of obviousness by the Patent Office, the applicant may attack the Patent Office’s *prima facie* determination as improperly made out, present objective evidence

tending to support a conclusion of nonobviousness, or both. *In re Fritch*, 972 F.2d 1260, 1265, 23 U.S.P.Q.2d 1780, 1783 (Fed. Cir. 1992).

With respect to claims 6-8, 12, and 13, Applicant respectfully submits that the applied references cannot be combined nor modified to produce the claimed invention. Hence, the rejection of claims 6-8, 12, and 13 does not establish a *prima facie* case of obviousness based on the prior art. Therefore, the rejection of claims 6-8, 12, and 13 under 35 U.S.C. § 103(a) has been shown to be insupportable, and these claims are patentable over the applied references. Applicant requests that the rejection be withdrawn.

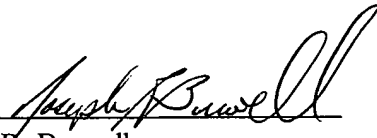
V. Conclusion

It is respectfully urged that the present patent application is patentable, and Applicant kindly requests a Notice of Allowance.

For any other outstanding matters or issues, the examiner is urged to call or fax the below-listed telephone numbers to expedite the prosecution and examination of this application.

DATE: July 13, 2001

Respectfully submitted,



Joseph R. Burwell
Reg. No. 44,468
ATTORNEY FOR APPLICANT

Law Office of Joseph R. Burwell
P.O. Box 28022
Austin, Texas 78755
(512) 502-9448 (voice)
(512) 597-1218 (fax)

APPENDIX

1. A method for deleting entries from a directory in which directory information is stored in a set of database tables, comprising the steps of:

responsive to a request to delete a directory entry, tagging the directory entry in a first table;
periodically searching for tagged entries in the first table during a cleanup process interval; and
deleting references to the tagged entries throughout the set of database tables.

2. The method as described in claim 1 wherein the directory entry is tagged by setting its creation time to a given value.

3. The method as described in claim 2 wherein the given value is a null value.

4. The method as described in claim 1, further including the steps of:
performing a search for directory entries that satisfy a search query; and
excluding tagged entries from search results that otherwise satisfy the search query.

5. The method as described in claim 4 wherein the step of excluding tagged entries includes modifying an SQL query to exclude rows having a null change creation.

6. The method as described in claim 1 wherein the directory is a Lightweight Directory Access Protocol (LDAP) directory service and the database tables are managed by a relational database management service.

7. The method as described in claim 1 wherein the first table is an entry table.

8. The method as described in claim 7 wherein the set of database tables includes at least one attribute table storing information about an attribute.

9. A method for deleting entries from a directory in which directory information is stored in a set of database tables, comprising the steps of:

responsive to a request to delete a directory entry, tagging the directory entry in a first table;
responsive to a search for directory entries that satisfy a search query, excluding tagged entries from search results that otherwise satisfy the search query;
periodically searching for tagged entries during a cleanup process interval; and
deleting references to the tagged entries throughout the set of database tables.

10. The method as described in claim 9 wherein the directory entry is tagged by setting its creation time to a given value.

11. The method as described in claim 10 wherein the given value is a null value.

12. The method as described in claim 9 wherein the first table is an entry table.
13. The method as described in claim 12 wherein the set of database tables includes at least one attribute table storing information about an attribute.
14. A method for searching a database from a directory service, comprising the steps of:
- responsive to a search for directory entries that satisfy a search query, excluding given entries from search results that otherwise satisfy the search query, wherein the given entries identify database entries that have been tagged for deletion; and
- returning the search results.
15. The method as described in claim 14 where in the directory service is a Lightweight Directory Access Protocol (LDAP) directory service and the database tables are managed by a relational database management service.

16. A computer program product in a computer-readable medium for deleting entries from a directory in which directory information is stored in a set of database tables, comprising:

means responsive to a request to delete a directory entry for tagging the directory entry in a first table;

means for periodically searching for tagged entries in the first table during a cleanup process interval; and

means for deleting references to the tagged entries throughout the set of database tables.

17. The computer program product as described in claim 16, further including:

means responsive to a search for directory entries that satisfy a search query for excluding tagged entries from search results that otherwise satisfy the search query.

18. The computer program product as described in claim 17 wherein the search query is a Lightweight Directory Access Protocol (LDAP) directory service query.

19. A directory service, comprising:

a directory organized as a naming hierarchy having a plurality of entries each represented by a unique identifier;

a relational database management system having a backing store for storing directory data in a set of database entries; and

means for deleting entries from the directory, comprising:

means responsive to a request to delete a directory entry for tagging the directory entry in a first table;

means for periodically searching for tagged entries in the first table during a cleanup process interval;

means for deleting references to the tagged entries throughout the set of database tables; and

means responsive to a search for directory entries that satisfy a search query for excluding tagged entries from search results that otherwise satisfy the search query.

20. The directory service as described in claim 19 wherein the directory is compliant with the Lightweight Directory Access Protocol (LDAP).